Visual Basic Language Reference

CreateObject Function (Visual Basic)

See Also Example

⊟ Collapse All Language Filter: Visual Basic

Creates and returns a reference to a COM object. **CreateObject** cannot be used to create instances of classes in Visual Basic unless those classes are explicitly exposed as COM components.

```
Public Shared Function CreateObject( _
   ByVal ProgId As String, _
   Optional ByVal ServerName As String = "" _
) As Object
```

Parameters

ProgId

Required. **String**. The program ID of the object to create.

ServerName

Optional. **String**. The name of the network server where the object will be created. If *ServerName* is an empty string (""), the local computer is used.

Exceptions

Exception type	Error number	Condition
Exception	429	*ProgId* not found or not supplied -or- *ServerName* fails the DnsValidateName function, most likely because it is longer than 63 characters or contains an invalid character.
Exception	462	Server is unavailable
FileNotFoundException	53	No object of the specified type exists

See the "Error number" column if you are upgrading Visual Basic 6.0 applications that use unstructured error handling. (You can compare the error number against the Number Property (Err Object).) However, when possible, you should consider replacing such error control with Structured Exception Handling Overview for Visual Basic.

Remarks

To create an instance of a COM component, assign the object returned by **CreateObject** to an object variable:

Copy Code

```
Sub CreateADODB()
   Dim adoApp As Object
   adoApp = CreateObject("ADODB.Connection")
End Sub
```

The type of object variable you use to store the returned object can affect your application's performance. Declaring an object variable with the **As Object** clause creates a variable that can contain a reference to any type of object. However, access to the object through that variable is *late-bound*, that is, the binding occurs when your program runs. There are many reasons you should avoid late binding, including slower application performance.

You can create an object variable that results in early binding—that is, binding when the program is compiled. To do so, add a reference to the type library for your object from the **COM** tab of the **Add Reference** dialog box on the **Project** menu. Then declare the object variable of the specific type of your object. In most cases, it is more efficient to use the **Dim** statement and a primary interop assembly to create objects than it is to use the **CreateObject** function.

Interacting with Unmanaged Code

Another issue is that COM objects use unmanaged code — code without the benefit of the common language runtime. There is a fair degree of complexity involved in mixing the managed code of Visual Basic with unmanaged code from COM. When you add a reference to a COM object, Visual Basic searches for a primary interop assembly (PIA) for that library; if it finds one, then it uses it. If it does not find a PIA, then it creates an interoperability assembly that contains local interoperability classes for each class in the COM library. For more information, see COM Interoperability in .NET Framework Applications.

You should generally use strongly bound objects and primary interop assemblies whenever possible. The examples below use the **CreateObject** function with Microsoft Office objects for demonstration purposes only. However, these objects are easier to use and more reliable when used with the appropriate primary interop assembly.

Creating an Object on a Remote Computer

You can create an object on a remote networked computer by passing the name of the computer to the *ServerName* argument of the **CreateObject** function. That name is the same as the Machine Name portion of a share name: for a share named "\\MyServer\Public," *ServerName* is "MyServer."

Note

Refer to COM documentation (see Microsoft Developer Network) for additional information on making an application accessible on a remote networked computer. You may need to add a registry key for your application.

The following code returns the version number of an instance of Excel running on a remote computer named `MyServer`:

Copy Code

```
Sub CreateRemoteExcelObj()
    Dim xlApp As Object
    ' Replace string "\\MyServer" with name of the remote computer.
    xlApp = CreateObject("Excel.Application", "\\MyServer")
    MsgBox(xlApp.Version)
End Sub
```

If the remote server name is incorrect, or if it is unavailable, a run-time error occurs.

Note

Use **CreateObject** when there is no current instance of the object. If an instance of the object is already running, a new instance is started, and an object of the specified type is created. To use the current instance, or to start the application and have it load a file, use the **GetObject** function. If an object has registered itself as a single-instance object, only one instance of the object is created, no matter how many times **CreateObject** is executed.

Creating Framework Objects

You can use the **CreateObject** function only to create a COM object. While there is no exact equivalent mechanism for creating a .NET Framework object, the Activator in the System namespace contains methods to create local or remote objects. In particular, the CreateInstance method or the CreateInstanceFrom method might be useful.

Security Note

The **CreateObject** function requires unmanaged code permission, which might affect its execution in partial-trust situations. For more information, see SecurityPermission and Code Access Permissions.

Example

The following example uses the **CreateObject** function to create a Microsoft Excel worksheet and saves the worksheet to a file. To use this example, Excel must be installed on the computer where this program runs. Also, you must add a reference to the type library from the **COM** tab of the **Add Reference** dialog box on the **Project** menu. The name of the type library varies depending on the version of Excel installed on your computer. For example, the type library for Microsoft Excel 2002 is named **Microsoft Excel 10.0 Object Library**.

Visual Basic Copy Code

```
Sub TestExcel()
    Dim xlApp As Microsoft.Office.Interop.Excel.Application
    Dim xlBook As Microsoft.Office.Interop.Excel.Workbook
    Dim xlSheet As Microsoft.Office.Interop.Excel.Worksheet

    xlApp = CType(CreateObject("Excel.Application"), _
                Microsoft.Office.Interop.Excel.Application)
    xlBook = CType(xlApp.Workbooks.Add, _
                Microsoft.Office.Interop.Excel.Workbook)
    xlSheet = CType(xlBook.Worksheets(1), _
                Microsoft.Office.Interop.Excel.Worksheet)

    ' The following statement puts text in the second row of the sheet.
    xlSheet.Cells(2, 2) = "This is column B row 2"
    ' The following statement shows the sheet.
    xlSheet.Application.Visible = True
    ' The following statement saves the sheet to the C:\Test.xls directory.
    xlSheet.SaveAs("C:\Test.xls")
    ' Optionally, you can call xlApp.Quit to close the workbook.
End Sub
```

Smart Device Developer Notes

This function is not supported.

Requirements

Namespace: Microsoft.VisualBasic

Module: Interaction

Assembly: Visual Basic Runtime Library (in Microsoft.VisualBasic.dll)

See Also

Reference

GetObject Function (Visual Basic)
Dim Statement (Visual Basic)
Declare Statement
Exception
FileNotFoundException
Activator
CreateInstance
CreateInstanceFrom

Other Resources

COM Interoperability in .NET Framework Applications
Interoperating with Unmanaged Code

To make a suggestion or report a bug about Help or another feature of this product, go to the feedback site.